Computer Systems

Exercise Session Week 9

Exercise Session Week 9

- **Last Sheet: Advanced questions**
- Recap
  - Chapter 17 – Byzantine Agreement
  - Chapter 18 – Broadcast & Shared Coins
- Next Sheet: Quiz questions

1.3 Improving Paxos

- Use different initial ticket numbers
- Servers reply to $\mathrm{ask}(t)$ with $\mathrm{nack}(T_{max})$ if t $< T_{max}$. (Instead of ignoring the message)
- When receiving a $\mathrm{nack}(T_{max})$, clients will try ticket $T_{max} + 1$ next.

➢ **EduApp:**

a) Does this improve runtime?

b) We now use a different approach: We add a wait time between 2 consecutive ask messages. How can you improve runtime like this? Try to not slow down an individual client when it is alone.

2.3 Consensus with bandwidth limitations

- No node/edge crashes
- Messages transmitted reliably and arrive after 1 time unit
- Every node can send 1 message with 1 value to 1 neighbour per time unit

➢ **EduApp:**

a) Develop consensus algorithm. What's the runtime?

b) All nodes must learn input value of all nodes. Show that runtime is at least $n-1$.

Exercise Session Week 9

- Last Sheet: Advanced questions
- Recap
  - **Chapter 17 – Byzantine Agreement**
  - Chapter 18 – Broadcast & Shared Coins
- Next Sheet: Quiz questions

Byzantine nodes

- Node which has (almost) arbitrary behavior
- It can:
  - Decide not to send messages
  - Sending different messages to different nodes
  - Sending wrong messages
  - Lie about input value
- It can't:
  - Forge an incorrect sender address
  - Forge signatures or beat cryptographic assumptions
- If an algorithm works with f byzantine nodes, it is f-resilient

Algorithm 15.13 Paxos

Client (Proposer) | **Server (Acceptor)**

Initialization ....

Client:

c ◁ *command to execute*
$t = 0$ ◁ *ticket number to try*

Server:

$T_{\text{max}} = 0$ ◁ *largest issued ticket*

$C = \bot$ ◁ *stored command*
$T_{\text{store}} = 0$ ◁ *ticket used to store C*

Phase 1 ....

Client:

$t = t + 1$
Ask all servers for ticket t

Server:

if $t > T_{\text{max}}$ **then**
  $T_{\text{max}} = t$
  Answer with `ok`(T_{store}, C)
end if

Phase 2 ....

Client:

if a majority answers `ok` **then**
  Pick (T_{store}, C) with largest T_{store}
  if $T_{\text{store}} > 0$ **then**
    $c = C$
  end if
  Send `propose`(t, c) to same majority
end if

Server:

if $t = T_{\text{max}}$ **then**
  $C = c$
  $T_{\text{store}} = t$
  Answer `success`
end if

Phase 3 ....

Client:

if a majority answers `success` **then**
  Send `execute`(c) to every server
end if

Algorithm 16.15 Randomized Consensus (assuming $f < n/2$)

$v_i \in \{0, 1\}$ ◁ input bit
round = 1
while true **do**
  Broadcast `myValue`(v_i, round)

  Propose

  Wait until a majority of `myValue` messages of current round arrived
  if all messages contain the same value v **then**
    Broadcast `propose`(v, round)
  else
    Broadcast `propose`($\perp$, round)
  end if

  Vote

  Wait until a majority of `propose` messages of current round arrived
  if all messages propose the same value v **then**
    Broadcast `myValue`(v, round + 1)
    Broadcast `propose`(v, round + 1)
    Decide for v and terminate
  else if there is at least one proposal for v **then**
    $v_i = v$
  else
    Choose v_i randomly, with $Pr[v_i = 0] = Pr[v_i = 1] = 1/2$
  end if
  round = round + 1
end while

Different Validities

- **Any-input validity:**
  - The decision value must be input of any node
  - That includes byzantine nodes, might not make sense
- **Correct-input validity:**
  - The decision value must be input of a correct node
  - Difficult because byzantine node could behave like normal one just with different value
- **All-same validity:**
  - If all correct nodes start with the same value, the decision must be that value
- **Median validity:**
  - If input values are orderable, byzantine outliers can be prevented by agreeing on a value close to the median value of the correct nodes
  - The median is the value separating the upper half from the lower half of a data sample.

Byzantine agreement in the synchronous model

- **Assumption:** nodes operate in **synchronous** rounds. In each round, each node may send a message to each other node, receive the message by other nodes and do some computation.
  - -> runtime is easy, since it is only the number of rounds

King Algorithm (synchronous byzantine agreement)

Idea:

- Once all correct nodes have the same value, we can easily make a decision.
  - ➢ We receive at least $n - f$ times same value
- So let's have one correct node decide on the value and broadcast it. Then all nodes choose it.

Problem:

- What if the "correct node" turns byzantine.
  - ➢ Have $f + 1$ such "king nodes"!

Algorithm 11.14 King Algorithm (for $f < n/3$)

x = my input value
for phase = 1 to $f + 1$ **do**
    Round 1
    Broadcast `value`(x)
    Round 2
    if some `value`(y) received at least $n - f$ times **then**
        Broadcast `propose`(y)
    end if
    if some `propose`(z) received more than f times **then**
        $x = z$
    end if
    Round 3
    Let node v_i be the predefined king of this phase i
    The king v_i broadcasts its current value w
    if received strictly less than $n - f$ `propose`(y) **then**
        $x = w$
    end if
end for

King Algorithm (synchronous byzantine agreement)

Idea:
- Once all correct nodes have the same value, we can easily make a decision.
  - ➢ We receive at least $n - f$ times same value
- So let's have one correct node (king) decide on the value and broadcast it. Then all nodes choose it.

Problem:
- What if the king turns byzantine.
  - ➢ Have $f + 1$ kings!

Algorithm 11.14 King Algorithm (for $f < n/3$)

x = my input value
for phase = 1 to $f + 1$ **do**

Do until at least one correct king

Round 1

Broadcast value(x)

Send out own value

Round 2

if some value(y) received at least $n - f$ times **then**
Broadcast propose(y)
end if

If we know that there's a majority in the correct nodes, propose that value. We always know that there's a majority, if all correct nodes have same value.

if some propose(z) received more than f times **then**
$x = z$
end if

If at least one correct node knows of a "correct majority", join the majority.

Round 3

Let node v_i be the predefined king of this phase i
The king v_i broadcasts its current value w

King of this phase broadcasts its value

if received strictly less than $n - f$ propose(y) **then**
$x = w$
end if
end for

If not all correct nodes already have the same value, then choose the king's value

King Algorithm (synchronous byzantine agreement)

- **Does it solve byzantine agreement?**
  - **Validity:** All same validity!
  - **Agreement:** They agree at least after the first correct king.
  - **Termination:** After (f+1)*3 rounds

Algorithm 11.14 King Algorithm (for $f < n/3$)

x = my input value
for phase = 1 to $f+1$ **do**
  Round 1
  Broadcast value(x)
  Round 2
  if some value(y) received at least $n - f$ times **then**
    Broadcast propose(y)
  end if
  if some propose(z) received more than f times **then**
    $x = z$
  end if
  Round 3
  Let node v_i be the predefined king of this phase i
  The king v_i broadcasts its current value w
  if received strictly less than $n - f$ propose(y) **then**
    $x = w$
  end if
end for

Asynchronous Byzantine Agreement

Assumption: Messages do not need to arrive at the same time anymore. They have variable delays.

➢ We can use the <u>exact</u> same idea as when there are only crashes.

Algorithm 17.21 Asynchronous Byzantine Agreement (Ben-Or, for $f < n/10$)

$x_u \in \{0, 1\}$ ◁ input bit
round = 1 ◁ round
while true **do**
    Broadcast `propose`(x_u,round)
    Wait until $n - f$ `propose` messages of current round arrived
    if at least $n/2 + 3f + 1$ `propose` messages contain same value x **then**
        Broadcast `propose`(x,round + 1)
        Decide for x and terminate
    else if at least $n/2 + f + 1$ `propose` messages contain same value x **then**
        $x_u = x$
    else
        choose x_u randomly, with $Pr[x_u = 0] = Pr[x_u = 1] = 1/2$
    end if
    round = round + 1
end while

“Default”: Flip coin and broadcast value

Algorithm 16.15 Randomized Consensus (assuming $f < n/2$)

$v_i \in \{0, 1\}$ ◁ input bit
round $= 1$
while true **do**
    Broadcast `myValue`(v_i, round)

    Propose

    Wait until a majority of `myValue` messages of current round arrived
    if all messages contain the same value v **then**
        Broadcast `propose`(v, round)
    else
        Broadcast `propose`($\perp$, round)
    end if

    Vote

    Wait until a majority of `propose` messages of current round arrived
    if all messages propose the same value v **then**
        Broadcast `myValue`(v, round + 1)
        Broadcast `propose`(v, round + 1)
        Decide for v and terminate
    else if there is at least one proposal for v **then**
        $v_i = v$
    else
        Choose v_i randomly, with $Pr[v_i = 0] = Pr[v_i = 1] = 1/2$
    end if
    round = round + 1
end while

Algorithm 17.21 Asynchronous Byzantine Agreement (Ben-Or, for $f < n/10$)

$x_u \in \{0, 1\}$ ◁ input bit
round $= 1$ ◁ round
while true **do**
    Broadcast `propose`(x_u,round)
    Wait until $n - f$ `propose` messages of current round arrived
    if at least $n/2 + 3f + 1$ `propose` messages contain same value x **then**
        Broadcast `propose`(x,round + 1)
        Decide for x and terminate
    else if at least $n/2 + f + 1$ `propose` messages contain same value x **then**
        $x_u = x$
    else
        choose x_u randomly, with $Pr[x_u = 0] = Pr[x_u = 1] = 1/2$
    end if
    round = round + 1
end while

Wait for $n - f$ messages: Is there a majority? Joint it!

Algorithm 16.15 Randomized Consensus (assuming $f < n/2$)

$v_i \in \{0, 1\}$ ◁ input bit
round = 1
while true **do**
  Broadcast `myValue`(v_i, round)
  Propose
  Wait until a majority of `myValue` messages of current round arrived
  if all messages contain the same value v **then**
    Broadcast `propose`(v, round)
  else
    Broadcast `propose`($\perp$, round)
  end if
  Vote
  Wait until a majority of `propose` messages of current round arrived
  if all messages propose the same value v **then**
    Broadcast `myValue`(v, round + 1)
    Broadcast `propose`(v, round + 1)
    Decide for v and terminate
  else if there is at least one proposal for v **then**
    $v_i = v$
  else
    Choose v_i randomly, with $Pr[v_i = 0] = Pr[v_i = 1] = 1/2$
  end if
  round = round + 1
end while

Algorithm 17.21 Asynchronous Byzantine Agreement (Ben-Or, for $f < n/10$)

$x_u \in \{0, 1\}$ ◁ input bit
round = 1 ◁ round
while true **do**
  Broadcast `propose`(x_u,round)
  Wait until $n - f$ `propose` messages of current round arrived
  if at least $n/2 + 3f + 1$ `propose` messages contain same value x **then**
    Broadcast `propose`(x,round + 1)
    Decide for x and terminate
  else if at least $n/2 + f + 1$ `propose` messages contain same value x **then**
    $x_u = x$
  else
    choose x_u randomly, with $Pr[x_u = 0] = Pr[x_u = 1] = 1/2$
  end if
  round = round + 1
end while

Do all nodes know of the majority? Decide and terminate!

Algorithm 16.15 Randomized Consensus (assuming $f < n/2$)

```
v_i ∈ {0,1}                ◁ input bit
round = 1
while true do
  Broadcast myValue(v_i, round)
  Propose
  Wait until a majority of myValue messages of current round arrived
  if all messages contain the same value v then
    Broadcast propose(v, round)
  else
    Broadcast propose(⊥, round)
  end if
  Vote
  Wait until a majority of propose messages of current round arrived
  if all messages propose the same value v then
    Broadcast myValue(v, round + 1)
    Broadcast propose(v, round + 1)
    Decide for v and terminate
  else if there is at least one proposal for v then
    v_i = v
  else
    Choose v_i randomly, with Pr[v_i = 0] = Pr[v_i = 1] = 1/2
  end if
  round = round + 1
end while
```

Algorithm 17.21 Asynchronous Byzantine Agreement (Ben-Or, for $f < n/10$)

```
x_u ∈ {0,1}                ◁ input bit
round = 1                  ◁ round
while true do
  Broadcast propose(x_u,round)
  Wait until n − f propose messages of current round arrived
  if at least n/2 + 3f + 1 propose messages contain same value x then
    Broadcast propose(x,round + 1)
    Decide for x and terminate
  else if at least n/2 + f + 1 propose messages contain same value x then
    x_u = x
  else
    choose x_u randomly, with Pr[x_u = 0] = Pr[x_u = 1] = 1/2
  end if
  round = round + 1
end while
```

Asynchronous Byzantine Agreement

The two algorithms also have the same problem:

- They're slow! (Expected exponential runtime)

Algorithm 17.21 Asynchronous Byzantine Agreement (Ben-Or, for $f < n/10$)

```
x_u ∈ {0, 1}                ◁ input bit
round = 1                   ◁ round
while true do
    Broadcast propose(x_u,round)
    Wait until n − f propose messages of current round arrived
    if at least n/2 + 3f + 1 propose messages contain same value x then
        Broadcast propose(x,round + 1)
        Decide for x and terminate
    else if at least n/2 + f + 1 propose messages contain same value x then
        x_u = x
    else
        choose x_u randomly, with Pr[x_u = 0] = Pr[x_u = 1] = 1/2
    end if
    round = round + 1
end while
```

Asynchronous Byzantine Agreement

The two algorithms also have the same problem:

- They're slow! (In expectation exponential runtime)

But we can use the same trick to improve on that:

- Shared coin / bitstring!
- But: If byzantine nodes know next round's bit, they can exploit that and the algorithm might never terminate. (See Theorem 17.29)

Algorithm 17.21 Asynchronous Byzantine Agreement (Ben-Or, for $f < n/10$)

$x_u \in \{0, 1\}$ ◁ input bit
round = 1 ◁ round
while true **do**
  Broadcast **propose**(x_u,round)
  Wait until $n - f$ **propose** messages of current round arrived
  if at least $n/2 + 3f + 1$ **propose** messages contain same value x **then**
    Broadcast **propose**(x,round + 1)
    Decide for x and terminate
  else if at least $n/2 + f + 1$ **propose** messages contain same value x **then**
    $x_u = x$
  If no popular value, look at bitstring
  end if
  round = round + 1
end while

Exercise Session Week 9

- Last Sheet: Advanced questions
- Recap
  - Chapter 17 – Byzantine Agreement
  - **Chapter 18 – Broadcast & Shared Coins**
- Next Sheet: Quiz questions

Secret can only be unveiled with cooperation of t nodes

Algorithm 18.22 (t, n)-Threshold Secret Sharing

1: Input: A secret s, represented as a real number.

Secret distribution by dealer d

2: Generate $t-1$ random numbers $a_1, \ldots, a_{t-1} \in \mathbb{R}$

3: Obtain a polynomial p of degree $t-1$ with $p(x) = s + a_1 x + \cdots + a_{t-1} x^{t-1}$

4: Generate n distinct $x_1, \ldots, x_n \in \mathbb{R} \setminus \{0\}$

5: Distribute share $\texttt{msg}(x_1, p(x_1))_d$ to node v_1, ..., $\texttt{msg}(x_n, p(x_n))_d$ to node v_n

Secret recovery

6: Collect t shares $\texttt{msg}(x_u, p(x_u))_d$ from at least t nodes

7: Use Lagrange's interpolation formula to obtain $p(0) = s$

Generate a bit string

Algorithm 18.23 Preprocessing Step for Algorithm 18.24 (code for dealer d)

1: According to Algorithm 18.22, choose polynomial p of degree f
2: **for** $i = 1, \ldots, n$ **do**
3:     Choose coinflip c_i, where $c_i = 0$ with probability $1/2$, else $c_i = 1$
4:     Using Algorithm 18.22, generate n shares $(x_1^i, p(x_1^i)), \ldots, (x_n^i, p(x_n^i))$ for c_i
5: **end for**
6: Send shares $\mathtt{msg}(x_u^1, p(x_u^1))_d, \ldots, \mathtt{msg}(x_u^n, p(x_u^n))_d$ to node u

---

Byzantine nodes need at least one correct node to unveil next round's bit

Algorithm 18.24 Shared Coin using Secret Sharing (ith iteration)

```
Replace Line 12 in Algorithm 17.21 by
Request shares from at least f + 1 nodes
Using Algorithm 18.22, let c_i be the value reconstructed from the shares
return c_i
```

Exercise Session Week 9

- Last Sheet: Advanced questions
- Recap
  - Chapter 17 – Byzantine Agreement
  - Chapter 18 – Broadcast & Shared Coins
- **Next Sheet: Quiz questions**

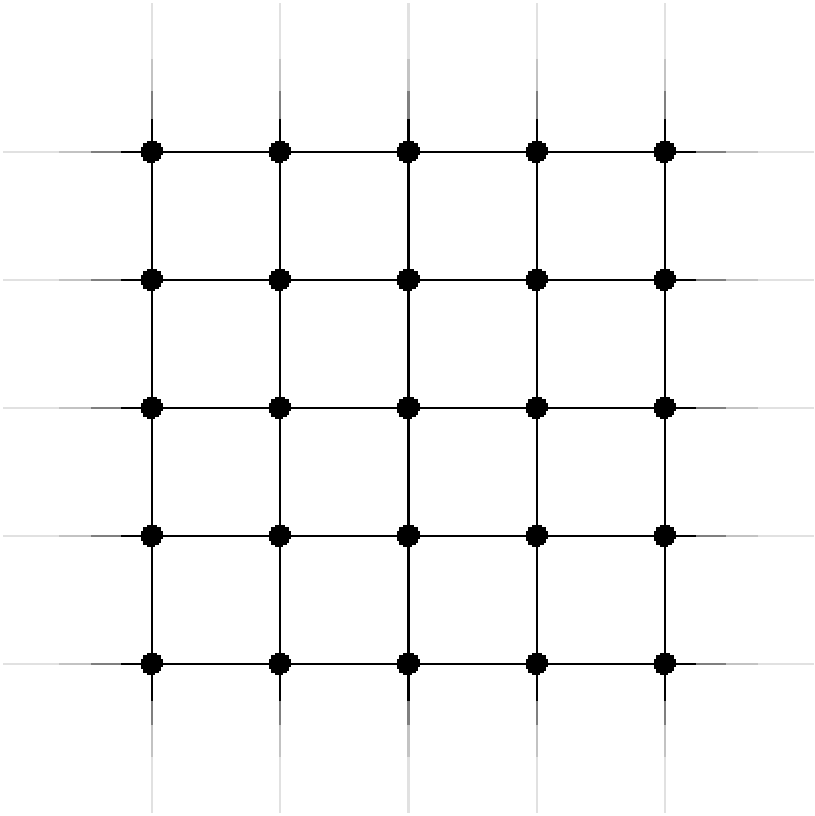
1.1 Synchronous consensus on a grid

- **EduApp:**

a) Consensus when w and h are known

b) Consensus when w and h are unknown

d) What's the smallest number of byzantine failures such that consensus might become impossible?



$$w \cdot h \gg w + h$$

2.1 What is the average?

7 nodes want to find the average of their inputs.

Inputs are: -3, -2, -1, 0, 1, 2, 3.

- **EduApp:**

a) What's the smallest number of failures (crash/byzantine) such that the task might become impossible?

b) If 2 nodes crash, in what range can the consensus value lie?

c) Additionally to the 7 correct ones, we have 2 byzantine nodes. In what range can the consensus value lie?